Adaptive Modelling and Control in Distributed Systems

Sophie Cerf, Mihaly Berekmeri, Nicolas Marchand, Sara Bouchenak, Bogdan Robu

Adaptive Modelling and Control in Distributed Systems

S. Cerf [1,2]
supervised by M. Berekmeri [1,2], N. Marchand [1,2], S. Bouchenak [3] and B. Robu [1,2]
[1] Univ. Grenoble Alpes, GIPSA-lab, F-38402 Grenoble, France
[2] CNRS, GIPSA-lab, F-38402 Grenoble, France
[3] LIRIS UMR 5205, INSA de Lyon, France
email: {sophie.cerf, mihaly.berekmeri, nicolas.marchand, bogdan.robu}@gipsa-lab.fr
sara.bouchenak@insa-lyon.fr

***Abstract*—Companies have growing amounts of data to store and to process. In response to these new processing challenges, Google developed MapReduce, a parallel programming paradigm which is becoming the major tool for BigData treatment. Even if MapReduce is used by most IT companies, ensuring its performances while minimising costs is a real challenge requiring a high level of expertise. Modelling and control of MapReduce have been developed in the last years, however there are still many problems caused by the software's high variability. To tackle the latter issue, this paper proposes an on-line model estimation algorithm for MapReduce systems. An adaptive control strategy is developed and implemented to guarantee response time performances under a concurrent workload while minimising resource use. Results have been validated using a 40 nodes MapReduce cluster under a data intensive Business Intelligence workload running on Grid5000, a french national cloud. The experiments show that the adaptive control algorithm manages to guarantee performances and low costs even in a highly variable environment.**

I. Introduction

Companies are producing more and more data every day: we are facing a BigData explosion which inevitably raises the challenges of their storage, analysis and processing. Traditional database techniques cannot be applied to treat this tremendous amount of data, which is generally not structured, therefore new paradigms are being developed to properly analyse and store BigData. Infrastructure as a Service clouds (IaaS) combined with parallel programming stand as a promising solution to tackle this issue. Their principle is to allocate on-demand hardware resources for different software applications according to requirements, thus providing almost unlimited storage and processing capacities.

The main challenge for parallel programming is to manage real time resources allocation. System performance, such as response time, need to be guaranteed while resource usage stays low in order to minimise both energetic and financial costs. This compromise is not easy to achieve as systems are highly variable over time: workloads are changing, shared infrastructures may cause concurrency issues, faults may arise, etc.

Currently, few automatic provisioning strategies have been implemented on cloud services. The most common method is a threshold based technique which adds or removes resources when thresholds are crossed, while keeping an eye on arising costs. Amazon developed this kind of strategy in its clouds EC2 [1] and EMR [2], where it is to the client to set the threshold used for provisioning. As users usually show few knowledge on the system operation, they cannot found an adequate threshold. Thus, this method cannot guarantee performances and does not minimise costs, automatic on-line solutions for resources provisioning that can guarantee performances need to be developed.

Control theory is getting more attention in computer science, as it enables to automatically deal with complex computing systems. Continuous control has been applied to monitor database server [3] or web service systems [4]. Discrete-time theory is also spreading, as reviewed in [5]. Recent publications propose linear dynamic modelling of a MapReduce system (a parallel programming tool) to capture its behaviour [6]. This predictive model is combined with a linear Proportional-Integral (PI) controller to meet performance criteria based on runtime. For simplicity reasons, this method is based on system linearisation around an operating point. As MapReduce behaviour is highly non-linear, performances can only be guaranteed close to this particular point, and the previous method shows it's limitations when the system is far away.

In the present paper, an adaptive control algorithm is applied to Hadoop, an open-source implementation of MapReduce. A real time estimator is used for system modelling to deal with its non-linearities. Based on these estimations, the correction part is also computed on-line. section 2 presents MapReduce and the experimental set-up used for validation. Section 3 sets out the on-line modelling and section 4 exposes the adaptive controller implemented. Preliminary results are developed in section 5. We conclude the paper with perspectives and future work.

II. Case Study : MapReduce

MapReduce is a parallel and distributed programming tool designed to handle input data partitioning, scheduling, machine failures, task distribution and load balancing [7]. It can capture, store and analyse requests on huge databases using a master-slave architecture. Developed in 2008 by Google,

Grid5000

Cluster	CPU	Memory	Storage	Network
40 nodes Grid5000	4 cores/CPU Intel 2.53GHz	15 GB	298 GB	Infiniband 20GB

MapReduce

Version & Distribution	Client Interaction
Apache Hadoop v1.1.2	Apache Hive

TABLE I. HARDWARE AND SOFTWARE CONFIGURATIONS

it has become the *de facto* tool for parallel programming. It is daily used by the biggest IT companies (Google, Yahoo, Facebook, etc. [7] [8]) to make numerous queries upon their databases. MapReduce is a popular but complex tool since many factors such as CPU and network skews or workload changes can influence jobs performances. Therefore, using it in a optimal way, that is to say minimise the cost while maximising the performance, is a real challenge. Since these disturbances occur during the runtime, optimal and robust resource configuration can only be found on-line.

For our test case, we use Hadoop, an open source version of MapReduce. To simulate jobs, we choose MRBS, a performance and dependability benchmark suite developed by [9] which can provide various workloads and inject several faults types. This benchmark suite has the advantage of simulating client interaction and being able to run concurrent jobs. We run Hadoop under a Business Intelligence workload on Grid5000, a nation-wide French cloud [10]. More details about the benchmark configuration can be found in Table I.

The experimental set-up has been developed according to the following procedure. First, SSH tunnels are set-up from local computer to remote on-line Grid5000 nodes. Then, MRBS generates the workload and the client interactions that will run on Hadoop. Meanwhile, actuators (which add and remove nodes) and sensors (which measure response time and the number of clients) are launch through Linux Bash scripts. Eventually, the Matlab implemented control strategy computes the number of nodes to set. Matlab is a high level technical computing language and an interactive environment for algorithm development, it is the standard tool in control theory. The control algorithm could have been written in any other programming language such as C or C++ without major changes. The Matlab implementation has been chosen for the experiments since it enables quick and easy development, implementing and testing of new control algorithms.

III. MAPREDUCE MODELLING

By system modelling we mean trying to capture its dynamic behaviour through mathematical relations between its relevant input and output signals. The objective is to have a mathematical model which can precisely predict the system behaviour, given the input data.

Since MapReduce is a very complex tool finding a detailed model is a difficult task as there is no physical equation that governs the system. Moreover MapReduce varies from one distribution to another, therefore we need

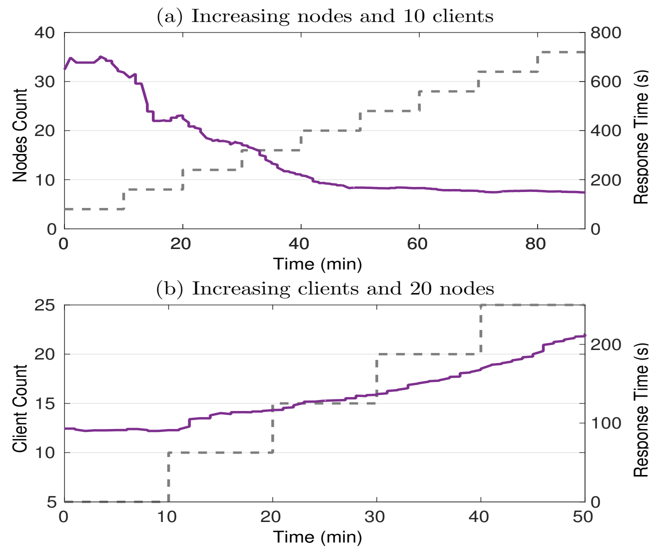

Fig. 1. MapReduce linearity analysis: response time with different input scenario

a model that is compatible with all versions. We decided to adopt a black box modelling approach, that is to say we capture relations between input and output signals only using experimental data. Control inputs and outputs have to be carefully chosen, therefore we analysed parameters that significantly impacted the system regardless the MapReduce version used. Justification of this choice can be found in [6], in the end we took our control input as the number of nodes, the number of clients is considered as an input disturbance and the control output is the jobs runtime.

Fig. 1 (a) presents the behaviour of MapReduce for a fixed number of clients and an increasing number of nodes. The non linearities of the system are highlighted by the three stages of the curve: a exponential decrease, a linear phase and a saturation. Even in the linear phase the model varies according to the chosen operating point and we choose to deal with this issue using adaptive algorithms. A linear model developed around a initial operating point (20 nodes and 10 clients) has been developed in [6] and is recalled in Fig. 2.

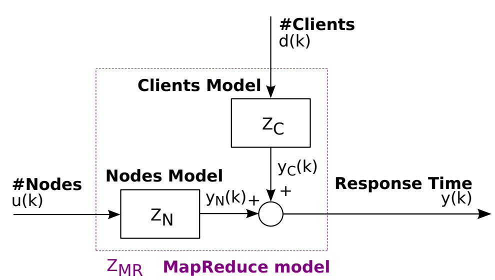

Fig. 2. MapReduce Model Schema

The relations between the number of nodes, the number of clients and the response time are captured by the equation (1) sampled at $T_s = 30s$, which parameters have been established in [6] around the (20 nodes, 10 clients) point. As we want our model to be valid over a wider range, we decided to keep the equation (1) structure but re-compute its parameters on-line to fit the real system. As most non linearities physically comes from nodes changes (see Fig. 1), we decided to update only a_0, b_0 and b_1. We supposed the clients to response time relation constant over time as Fig. 1 (b) shows an almost linear curve, therefore we used the parameters values identified by [6]: $\alpha_0 = -0.7915$, $\beta_0 = \beta_1 = 1.0716$. The delays in equation (1) are also considered as fixed as they are caused by the network.

$$\begin{aligned} y(k) &= y_N(k) + y_C(k) \\ y_N(k) &= -a_0 y_N(k-1) + b_1 u(k-5) + b_0 u(k-6) \\ y_C(k) &= -\alpha_0 y_C(k-1) + \beta_1 d(k-8) + \beta_0 d(k-9) \end{aligned} \tag{1}$$

We used a recursive least square estimator (RLSE) [11] to on-line update our model parameters. RLSE recursively optimises the model parameters by minimising a least square cost function between the measured response time and the response time estimated from equation (1) with the varying parameters a_0, b_0 and b_1. To compute new parameters the estimator requires input and output data (#Nodes,#Clients and Response time), past estimations and the chosen model structure. We choose this method, beyond being the most widely used estimator, because it is a well known algorithm that converges extremely fast [11]. As the system is varying over time, we use a combination of two different RLSE variants. If the estimation error, which is the difference between the measured output and the simulated one based on the estimated model, is higher than a certain threshold thd, we use RLSE with a exponential forgetting factor λ [11], as this method converges quickly. Else, if the estimation error is lower than the threshold thd we use the internal RLSE parameter reset (with the reset period T_R) since this method is more precise [11].

IV. MapReduce Control

Control theory aims at designing an autonomous control strategy which, based on present and past data of the system, computes an input signal to make the system act as we required. Though, the control algorithm is an equation between the error e (difference between the system output y and our desired value y_r) and the system input u (the number of nodes in our case), see Fig. 3.

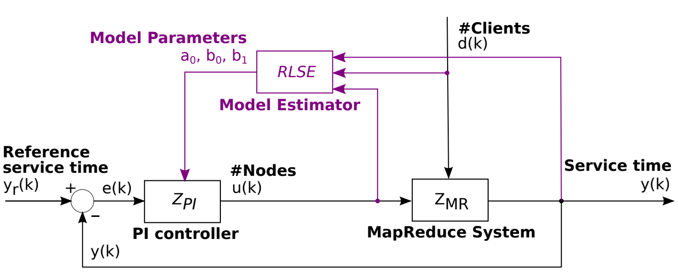

Fig. 3. Estimation and control loop scheme

It is very important to state that our control algorithm does not change the software itself, it is not intrusive and runs alongside the system. Our objective is to guarantee response time to a certain threshold while minimising resource usage on a wide range around the operating point (20 nodes, 10 clients), and robustness to any unexpected change in the system behaviour. Ensuring these specifications is a real challenge, particularly due to the large actuation delays of the system (more than 150s) and the quantization of the control input, which should always be a positive integer as it is the number of nodes.

Different control algorithms are defined by control theory, some are very complex but simple ones can prove to be sufficient. We choose to implement a Proportional-Integral control (see eq. (2)) as it is fast to compute, it efficiently follows specifications and presents good properties for disturbance rejection.

$$u(k) = u(k-1) + (K_p + K_i)e(k) + K_i e(k-1) \tag{2}$$

To compute the control parameters we use the model parameters, according to Astrom Hagglund relation (3) [12] of known PI design methods. As model parameters are estimated on line, the control equation has new parameters each sampling time:

$$K_p = 0.14\,\frac{1-a_0}{b_1+b_2}, \quad K_i = \frac{K_p}{11.5} \tag{3}$$

V. Preliminary Results

The model estimator presented in this article has been implemented and tested in real time on-line experiments on Grid5000 clusters using the values reported in Table II, results are shown in Fig. V.

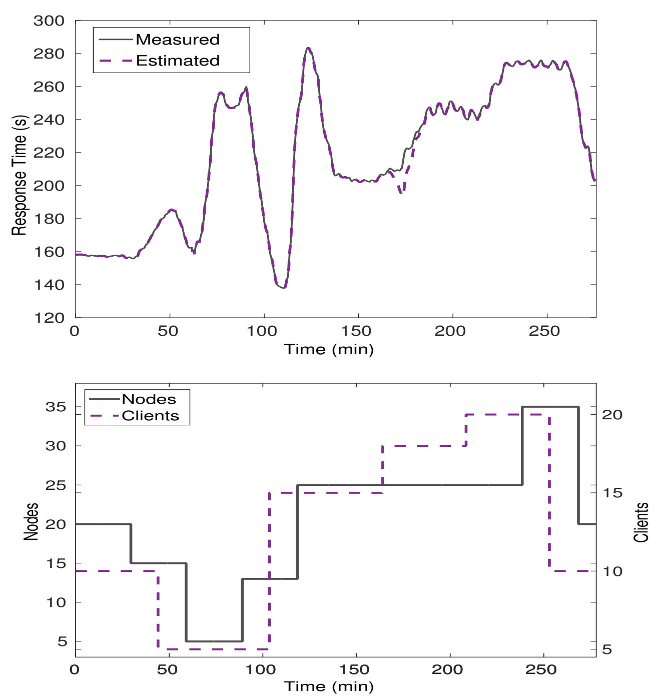

Fig. 4. Evaluating the model estimation accuracy

Forgetting factor	$\lambda = 0.6$
Reset Period	$T_R = T_S$
Threshold	$thd = 0.1y$

TABLE II. HARDWARE AND SOFTWARE CONFIGURATIONS

Regardless clients or nodes variations, the estimator manages to fit the system output, with some divergences that our combination of different RLSE algorithms well manages to make precisely converge again. The estimation is accurate even when input signals vary far from the operating point, that is to say when the system shows significant non-linearities (see Fig. 1), therefore our modelling is highly robust. Another interesting point is that the proposed estimation algorithm does not need a learning phase or learning data, it can be started at any time and it converges almost instantaneously.

The results of the implementation of our adaptive PI control algorithm for a 50% rise in the number of clients can be seen in Fig. 5. The response time rises for a few minutes whereas our control manages to drag it close to its original value, by slowly increasing the number of nodes, which implies low energetic and financial costs. However, the overall settling time is quite slow, as it lasts for more than 100 sampling times (50 minutes).

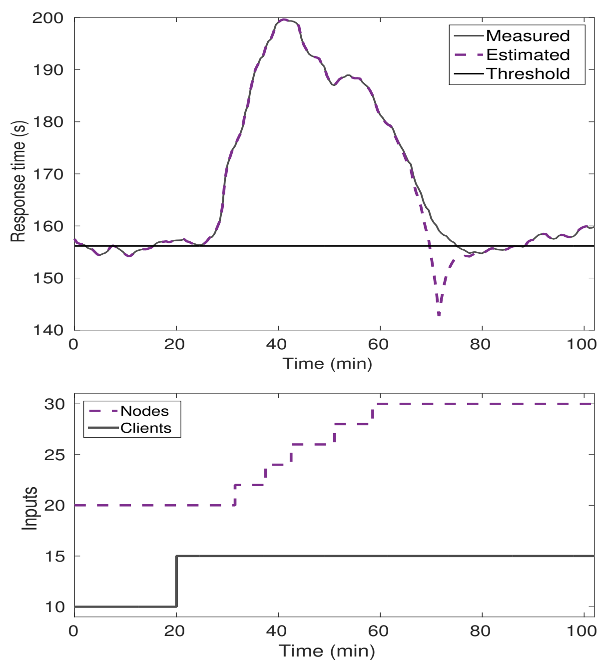

Fig. 5. Corrector validation by a 50% client increase

VI. CONCLUSION & PERSPECTIVES

This paper presents the conception, implementation and validation on an on-line adaptive modelling of MapReduce and a PI control strategy to guarantee its performances. We choose a black box model structure and a combination of recursive least square algorithms to estimate the model parameters in real time. Based on these estimations, the parameters of the discrete PI controller is also computed on line. Our experiments on Grid5000 show that our control framework successfully fulfil our specifications, as modelling fits the systems behaviour and control keeps performances at a certain threshold, while limiting resource usage.

Future work will explore the following directions:

- Validation process of the presented estimator and control in an on-line cloud, such as Amazon EC2.
- Control theory provides other controller structures, as feed-forward controller which cancel measured disturbances effects before they impact the system. Its implementation coupled with on-line estimation is considered.
- Minimising changes in the input signal can be achieve using event-based techniques, which computes a new control input only when the system outputs change significantly.
- Add new inputs and outputs to the model such as availability, the maximum number of clients allowed or the system throughput.
- As our system shows substantial non linearities, the non linear field of control theory can be explored to achieve better performances.

REFERENCES

[1] "AWS | Amazon Elastic Compute Cloud (EC2) Hbergement volutif dans le cloud." [Online]. Available: //aws.amazon.com/fr/ec2/

[2] "AWS | Amazon Elastic MapReduce (EMR) | Oprations Hadoop MapReduce dans le cloud." [Online]. Available: //aws.amazon.com/fr/elasticmapreduce/

[3] L. Malrait, N. Marchand, and S. Bouchenak, "Modeling and control of server systems: Application to database systems," in *Control Conference (ECC), 2009 European*, Aug. 2009, pp. 2960–2965.

[4] C. Poussot-Vassal, M. Tanelli, and M. Lovera, "Linear Parametrically Varying MPC for combined Quality of Service and energy management in Web service systems," in *American Control Conference (ACC), 2010*, Jun. 2010, pp. 3106–3111.

[5] E. Rutten, J. Buisson, G. D. ans F. de Lamotte, J.-F. Diguet, N. Marchand, and D. Simon, "Control of autonomic computing systems," 2013, submitted to ACM Computing Survey.

[6] M. Berekmery, D. Serrano, S. Bouchenak, N. Marchand, and B. Robu, "A Control Approach for Performance of Big Data Systems," in *19th IFAC World Congress (IFAC WC 2014)*, vol. 19, no. 1, Le Cap, South Africa, Aug. 2014.

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[8] Y. Shen, *Enabling the New Era of Cloud Computing: Data Security, Transfer, and Management: Data Security, Transfer, and Management*. IGI Global, 2013.

[9] A. Sangroya, D. Serrano, and S. Bouchenak, "Benchmarking dependability of mapreduce systems," in *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, Oct 2012, pp. 21–30.

[10] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard, "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, ser. GRID '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 99–106.

[11] K. J. Astrom and B. Wittenmark, *Adaptive Control: Second Edition*. Courier Corporation, Apr. 2013.

[12] A. O'Dwyer, *Handbook of PI and PID controller tuning rules*. World Scientific, 2009, vol. 57.